RL-TR-96-224
Final Technical Report
February 1997

OMARS EXTENSIONS: MULTIPLE ARCHITECTURES AND REAL-TIME CONSTRAINTS

Purdue University

John K. Antonio

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

Rome Laboratory
Air Force Materiel Command
Rome, New York

This report has been reviewed by the Rome Laboratory Public Affairs Office (PA) and is releasable to the National Technical Information Service (NTIS). At NTIS it will be releasable to the general public, including foreign nations.

RL-TR-96-224 has been reviewed and is approved for publication.

APPROVED:

RICHARD C. METZGER
Project Engineer

FOR THE COMMANDER:

JOHN A. GRANIERO, Chief Scientist
Command, Control, & Communications Directorate

If your address has changed or if you wish to be removed from the Rome Laboratory mailing list, or if the addressee is no longer employed by your organization, please notify RL/C3CB, 525 Brooks Road, Rome, NY 13441-4505. This will assist us in maintaining a current mailing list.

Do not return copies of this report unless contractual obligations or notices on a specific document require that it be returned.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY (Leave Blank)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	February 1997	Final Mar 95 - Dec 95

4. TITLE AND SUBTITLE
OMARS EXTENSIONS: MULTIPLE ARCHITECTURES AND REAL-TIME CONSTRAINTS

5. FUNDING NUMBERS
C - F30602-95-C-0079
PE - 62702F
PR - 5581
TA - 18
WU - PL

6. AUTHOR(S)
John K. Antonio

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Purdue University
School of Electrical Engineering
1285 Electrical Engineering Building
West Lafayette, IN 49707-1285

8. PERFORMING ORGANIZATION REPORT NUMBER
N/A

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
Rome Laboratory/C3CB
525 Brooks Road
Rome NY 13441-4515

10. SPONSORING/MONITORING AGENCY REPORT NUMBER
RL-TR-96-224

11. SUPPLEMENTARY NOTES
Rome Laboratory Project Engineer: Richard C. Metzger/C3CB/(315) 330-7652

12a. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution unlimited.

12b. DISTRIBUTION CODE

13. ABSTRACT (Maximum 200 words)

The work presented in this report is the result of collaboration between Rome Laboratory, Purdue University and Texas Tec University. This work builds on and extends our previous effort in which the initial version of OMARS (Optimal Mapping Alternate Routing System) was developed. In the current effort, OMARS is extended for use with two parallel architectures: the nCUBE2 and the Intel Paragon. Modifications were made to every major subsystem of OMARS, and these changes are documented in this report. In addition to these functional modifications, the utility of OMARS is also demonstrated using measured execution times of an FFT algorithm, which is commonly found in real-time signal processing applications. Also included is an evaluation of the alternate routing capability of the nCUBE2.

14. SUBJECT TERMS
Parallel Computing Systems, Tasks, Subtasks, Statis Allocation, Routing Analysis

15. NUMBER OF PAGES
36

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	UL

TABLE OF CONTENTS

List of Figures

List of Tables

Executive Summary

The work presented in this report is the result of collaboration between Rome Laboratory and Texas Tech University. This work builds on and extends our previous effort in which the initial version of OMARS (Optimal Mapping Alternate Routing System) was developed. In the current effort, OMARS is extended for use with two parallel architectures: the nCUBE 2 and the Intel Paragon (the original version of OMARS was operational with the nCUBE 2 only). Modifications were made to every major subsystem of OMARS, and these changes are documented in this report. In addition to these functional modifications, the utility of OMARS is also demonstrated using measured execution times of an FFT algorithm, which is commonly found in real-time signal processing applications. Also included is an evaluation of the alternate routing capability of the nCUBE 2.

1 Background and Motivation for the Work

Much of the general background and motivation for this work is the same as that discussed in the report for the original effort, "Software Techniques for Balancing Computation & Communication in Parallel Systems," Rome Laboratory, Air Force Materiel Command, Griffiss Air Force Base, NY, Final Technical Report No. RL-TR-94-98, July 1994 [2]. For brevity, some of these general issues are only overviewed here as needed; the interested reader should refer to the original report for more details.

1.1 Mapping and Routing Performance Issues

In general, static mapping and routing choices do impact delivered performance on parallel systems. This fact is one of the key motivators for the OMARS project. To illustrate the impact of mapping choice, consider the simple example shown in Fig. 1. The upper-left portion of the figure shows an example task graph in which the eight vertices represent subtasks (S_0 through S_7) and the directed edges represent communications among the subtasks. So, for example, the edge from S_0 to S_4 indicates that S_0 must send a message to S_4 at some point during the execution of the task graph. Shown in the upper-right portion of the figure is an architecture graph. Here, the eight vertices (P_0 through P_7) denote processors of a parallel machine and the interconnecting edges are communication links; the particular interconnection shown is a hypercube.

The lower portion of Fig. 1 illustrates the effects of two different mappings of subtasks to processors (Mappings A and B). For Mapping A, note that the paths for the required communications generally require more than one communication link. Also, under the assumption that the communications are initiated at approximately the same time, there will be contention for some of the communication links. For instance, assuming the links are bi-directional, the link connecting S_4 to S_6 is used to establish two distinct communication paths. If a cut-through routing scheme is employed [6], then contention will occur on this link, causing a delay in the time required for one of the paths to be established. Shown in the lower-right portion of the figure is Mapping B. Clearly, this mapping makes more effective use of the available communication links for the given communication pattern associated with the example task graph. In this mapping, the required communication paths among all pairs of communicating subtasks are nonconflicting; each utilizes a distinct communication link of the hypercube architecture.

As shown in Fig. 2, for a fixed mapping choice, routing choices can also impact delivered performance. In the upper portion of the figure is a sample task graph having nine subtasks, and a nine processor architecture graph interconnected as a 3×3 planar mesh. The lower

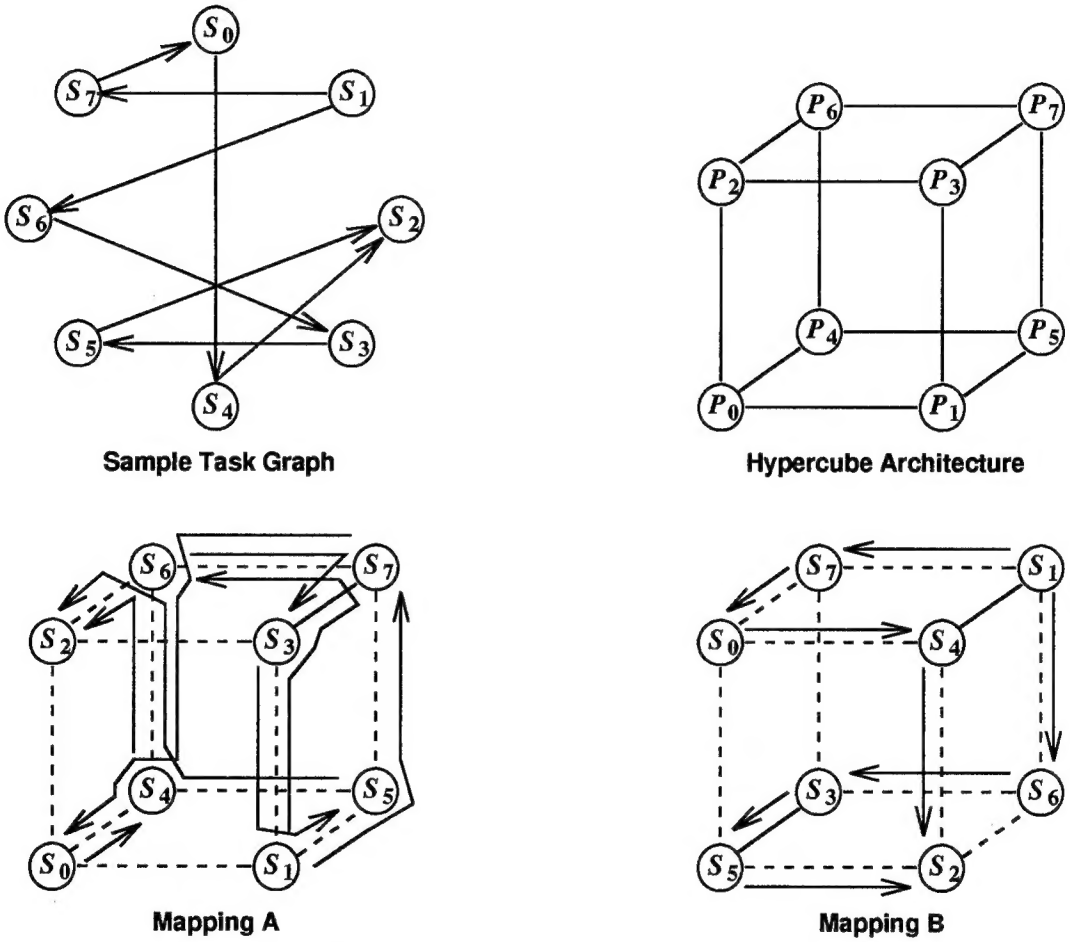

Figure 1: Example showing impact of mapping choice.

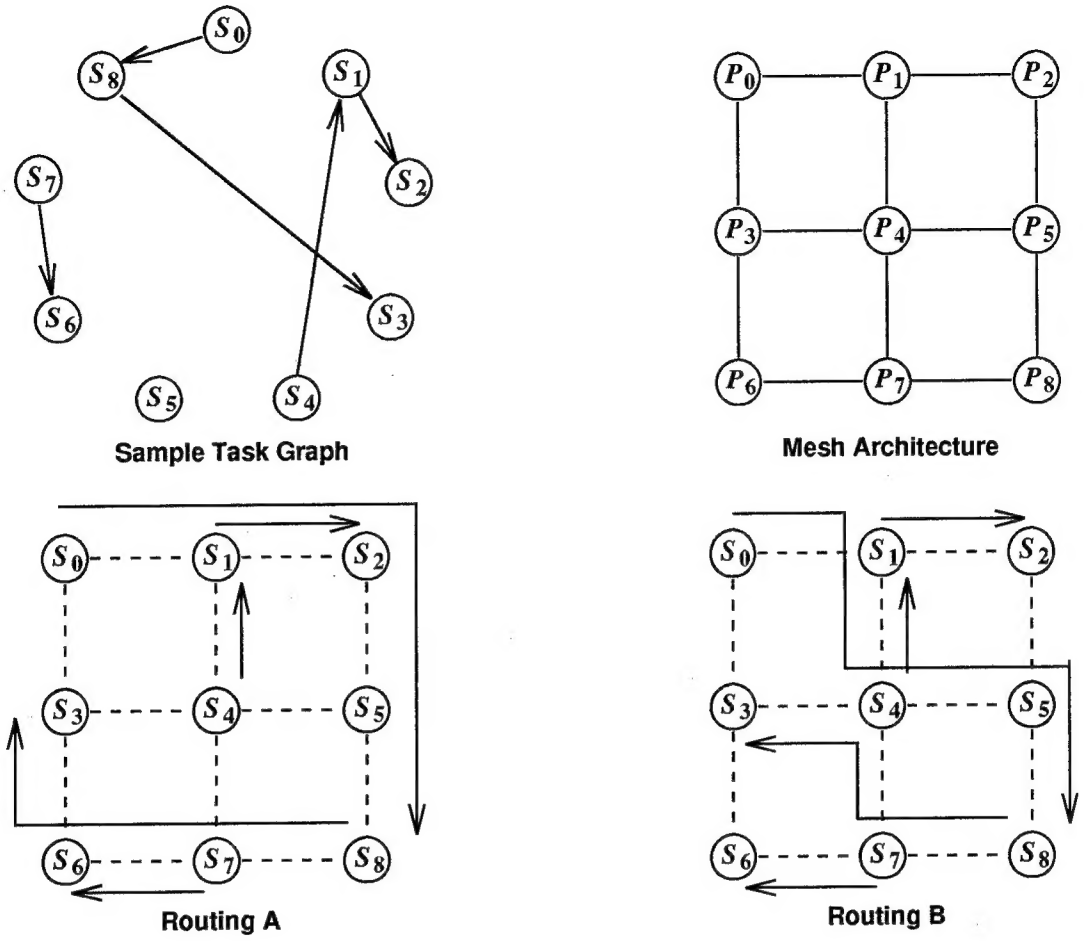

Figure 2: Example showing impact of routing choice.

portion shows the effect of two different routings for the same mapping choice. In Routing A, note that some of the communication links are utilized by two communication paths, e.g., the link connecting S_1 to S_2. However, in Mapping B, each (bi-directional) communication link is used by at most one path. Thus, utilizing Mapping B would generally result in less delay from link contentions.

OMARS is a tool that can help the user make mapping and routing choices that result in good performance. One of the goals of OMARS is to automate portions of the process by hiding the underlying details and complexity of solving mapping and routing problems from the user. However, flexibility is still provided to allow the user to decide which mapping/routing selection is ultimately implemented.

1.2 Software Evolution and Maintenance Issues

The functionality of complex parallel applications can evolve over time. When this occurs, the communication pattern among the subtasks of the application may also change. For

this reason, the problems of re-mapping and re-routing may need to be addressed in order to maintain peak performance. Thus, optimal mapping and routing are not necessarily "one-time" problems to be solved only during the initial phases of the parallel software development process. Instead, these problems may arise again as the functionality of the application is changed or enhanced. This introduces new software management problems, compared to the sequential software domain. A tool like OMARS could be employed to reduce the difficulty of addressing the mapping and routing problems throughout the lifecycle of a parallel application.

1.3 Overview of the Work

In OMARS, a parallel application program is instrumented and executed once to obtain a tracefile, which contains information about the communication pattern present in the parallel application program. This information is used to generate and display a task graph within OMARS. The target architecture is modeled using an appropriate architecture graph. With these models in place, the user can interactively select different mapping/routing techniques and view the results (both graphically and in terms of numerical metrics) of each selection. A detailed description of OMARS, including sample screen dumps, etc., can be found in [2] and [7].

A conceptual overview of the function of OMARS is shown in Fig. 3. Note that the input to OMARS is the tracefile. Based on this input, OMARS produces graphics and metrics (one for each candidate mapping/routing algorithm combination selected by the user) that aid the user in deciding how to best map/route subtasks/communications for the target architecture. The output of OMARS is a "taskfile," which contains the mapping and routing information produced by the user-selected algorithms within OMARS. This taskfile is applied when executing the application on the target architecture. The actual performance obtained can then be used a feedback information to the user to help in the tuning process of achieving the best possible mapping/routing choice. This is an important feedback link, as the performance metrics provided by OMARS are based on static analysis, and do not, therefore, always perfectly correlate with actual execution times (this issue is discussed further in Section 6).

The original version of OMARS was developed for operation with the nCUBE 2 machine, which is an MIMD machine with a hypercube interconnection network [8]. Thus, all of the mapping and routing techniques developed and implemented for the original version of OMARS were designed specifically for the hypercube architecture. Also, the tracing facilities were designed for the nCUBE 2.

The primary goal of this effort was to extend the operation of OMARS for use with

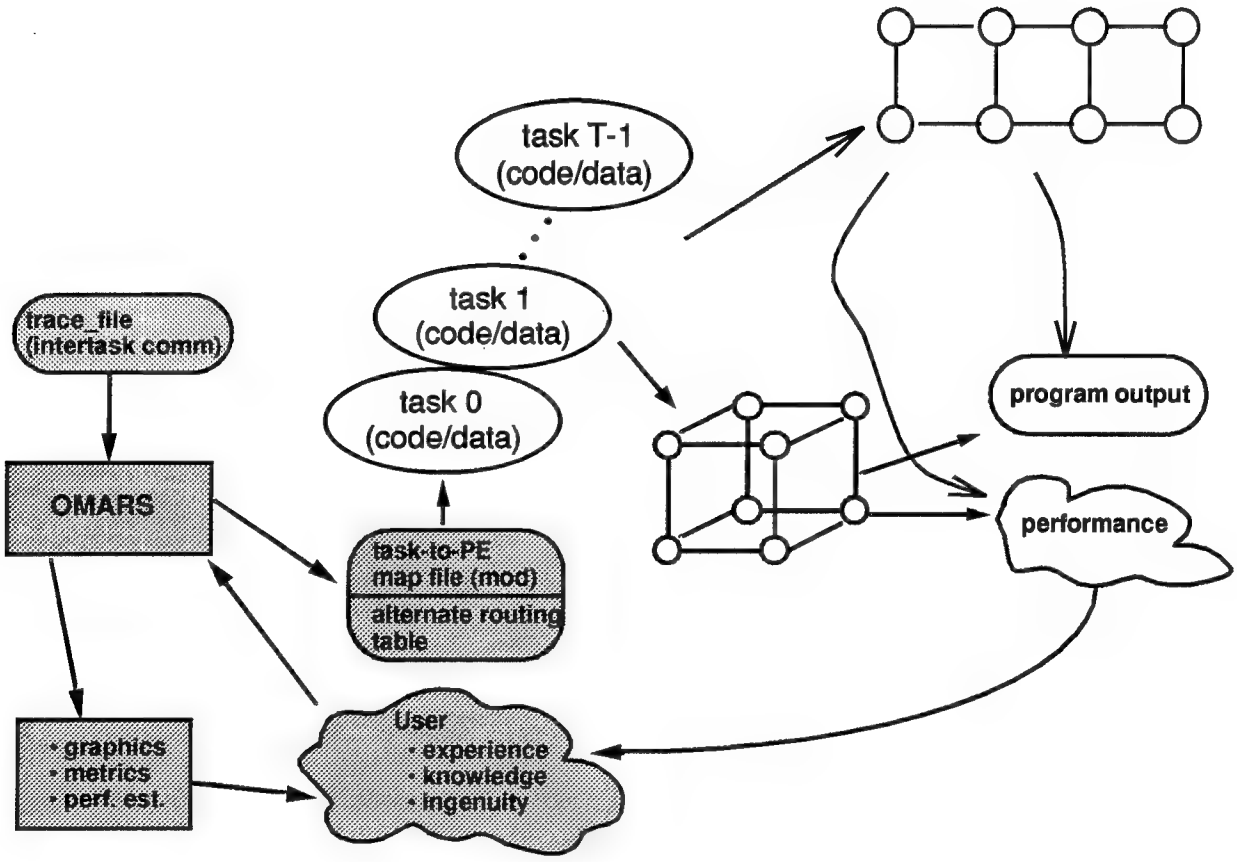

Figure 3: Conceptual overview of the function of OMARS.

another parallel architecture: the Intel Paragon, which is an MIMD machine with a mesh interconnection network. Thus, all of the mapping and routing techniques for the hypercube had to be modified (or, in some cases, completely re-designed) based on the mesh architecture of the Paragon. Also, the graphics had to be extended for displaying a mesh architecture graph. Numerous changes were also made to the user interface to enable the user to map an application, originally developed on one machine, onto the other machine.

The remainder of the report is organized as follows. In Section 2, the user interface modifications are described. In Section 3, the modifications associated with the tracing portion of OMARS are overviewed. The changes to the mapping and routing algorithms are described in Sections 4 and 5, respectively. In Section 6, timing results for an application study is conducted (based on an FFT algorithm) to demonstrate the impact that mapping can have on performance. Section 7 evaluates the alternate routing capability of the nCUBE 2.

2 User Interface Modifications

The most important interface to the user is made through the main menu. A comparison of the original and new versions of the main menu for OMARS is shown in Fig. 4. An overview of the specific user interface changes made are as follows:

- developed menu-driven tracefile selector interface;
- added "select target architecture" button;
- developed target architecture configuration interface (enables the "mixing" of tracefile source and target architectures);
- developed architecture window for Paragon (graphical view of mesh and metrics bar charts);
- developed an intuitive "grey-out" mechanism to prevent invalid menu selections (based on implementation of the valid state transition diagram); and
- added a "start over" button within the main menu.

Fig. 5 illustrates how the new user interface operates. Each of the four views of the menus shown represent a different "state" of operation during an OMARS session. The upper-left corner illustrates the main menu that is generated when OMARS is first started. For each state, only the "un-greyed" buttons are valid options. Thus, initially, the user can press only three buttons: "Read Tracefile", "Help", or "Quit OMARS". The arrows emanating from

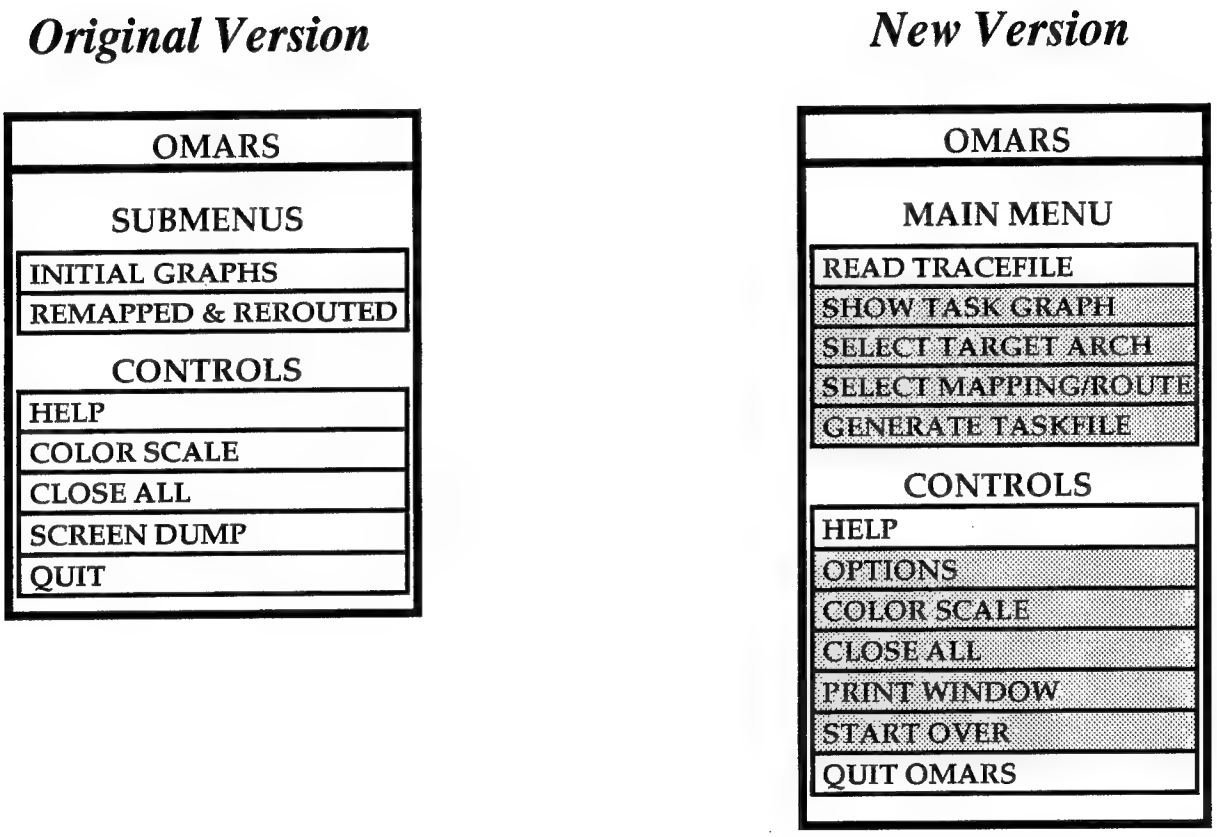

Figure 4: Comparison of original and new main menus.

each valid button describe the action and/or transition that is made. For example, if the "Read Tracefile" button is selected, then a tracefile pop-up window is generated. After the user makes selections from the tracefile pop-up window (not illustrated here), then the main menu transitions to the state shown in the upper-right corner of Fig. 5. If the "Help" button is selected, then a help index pop-up is generated. After receiving help, the menu returns to the same state from which the "Help" button was selected.

In a typical OMARS session, a user would first select the "Read Tracefile" button, and select a tracefile that appears in the tracefile pop-up window. Then, the user might select the "Show Task Graph" button to view the task graph associated with the selected tracefile. The task graph gives the user a graphical view of the application's communication pattern. Next, the user presses the "Select Target Architecture" button, which generates a target architecture pop-up. This pop-up allows the user to select either the nCUBE 2 or Intel Paragon architectures. After the basic architecture is decided, the user is further questioned to provide a configuration for the selected architecture. For the nCUBE 2, this means defining the dimension of the hypercube, for the Paragon, this means defining the "X" and "Y" dimensions of the mesh. Again, the details of these pop-up menus are not shown in Fig. 5; they are best seen through actual use of OMARS.

After selecting and configuring a target architecture, the user might then want to choose a mapping scheme and a routing scheme by pressing the "Select Mapping/Routing" button. This action results in the generation of pop-ups for selecting first the mapping algorithm, then the routing algorithm. Also generated is an architecture graph for this mapping/routing combination. The architecture graph gives a graphical view of the target architecture in which the vertices and edges are color-coded to indicate the utilizations of the processors and communication links, respectively. After the mapping and routing algorithms are selected, the main menu transitions to the final state shown in the lower-left corner of Fig. 5. From this state, the user can press the "Generate Taskfile" button, which creates a file containing the information for implementing the mapping and routing (produced by the selected algorithms) on the target architecture. Using this taskfile, the user can then execute the application on the target architecture as a means of tuning performance.

3 Tracefile Modifications

In the original version of OMARS, we developed our own instrumentation code for tracing the communications that occurred in an nCUBE 2 program. Initially, this code was ported to the Paragon. Although this "home-brew" instrumentation works, it does add a layer of complexity to the process of using OMARS. At about the mid-point of the effort (shortly

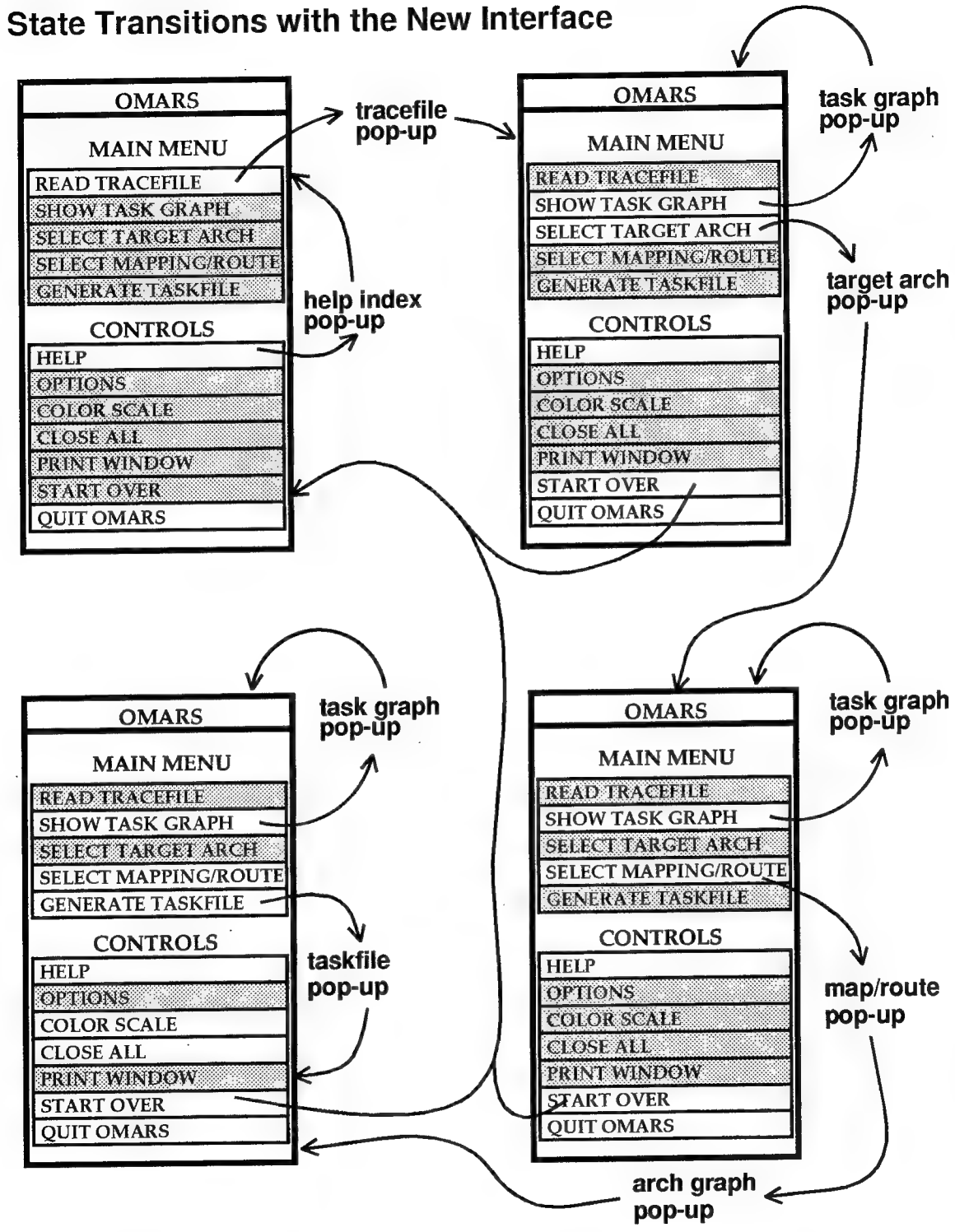

Figure 5: State transitions associated with the new user interface.

after porting our tracing facility to the Paragon), Intel introduced a software environment for the Paragon called IPD (Intel Parallel Debugger), which included a tracing facility. The tracefiles generated by IPD are very large, and contain very detailed information about the various function calls used by the application program including those associated with communications. Ultimately, we managed to "de-code" the format and content of the information contained in an IPD tracefile (which is a binary file), and were able to modify OMARS to accept IPD tracefiles.

Being able to read IPD tracefiles was an important enhancement made to OMARS, because it enables OMARS to leverage the use of a commercial software environment that is stable and supported by Intel. Thus, a Paragon user that is familiar with IPD can easily begin using OMARS.

Thus, OMARS accepts three types of tracefiles:

- nCUBE 2 tracefiles from our instrumentation;
- Paragon tracefiles from our instrumentation; and
- Paragon IPD tracefiles.

Another important feature that was added to OMARS was the ability to read a tracefile generated by one architecture with a particular configuration (e.g., the nCUBE 2 configured as a 3-dimensional hypercube) and target another architecture and/or configuration (e.g., the Paragon as a 3×4 mesh). An overview of the different possible combinations are given in Fig. 6. These choices add even more degrees of freedom in the utilization of OMARS. Thus, not only is the mapping/routing choice an option, but also the target architecture and its configuration. This enables the user to play "what if" games. For instance, suppose that performance requirements cannot be met using an mesh architecture. The user could then investigate the possibility of mapping the application onto a hypercube.

4 Mapping Algorithm Modifications

Some of the changes that were required in converting the original mapping algorithms (designed for hypercubes) to operate with the mesh topology are given in this section. Many of the details of this conversion process are not included here; instead, an overview of the conceptual changes required for selected algorithms is provided. The reader interested in the details of the implementation of a particular mapping algorithm should refer to the source code for that algorithm.

A key concept in the original version of the "greedy mapper" was to map subtasks to processors by considering the processors in a Gray code order [6]. Traversing processors

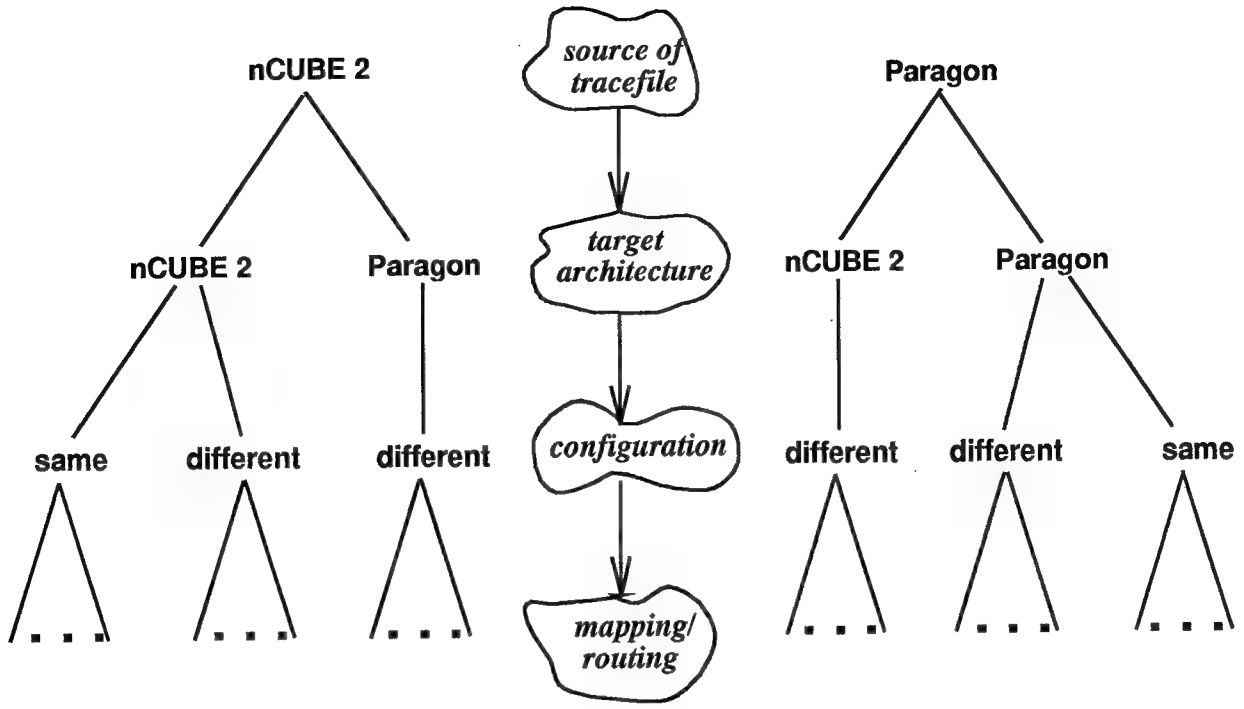

Figure 6: Overview of the number and types of combinations possible for a given tracefile to an ultimate mapping/routing selection.

of a hypercube in Gray code order is tantamount to traversing processors in an "nearest neighbor" order. Fig. 7 shows an example gray code traversal of a 3-dimensional hypercube. For the case of the mesh architecture, the traversal of the processors were done in what we define as a "snake order," as shown in Fig. 8. Traversing in a snake order on a mesh preserves the nearest neighbor property associated with the original hypercube version of the greedy mapper. Initial experiments done using the modified greedy mapper showed that reasonable mesh mappings can be obtained very quickly.

Another mapping algorithm that was modified was the original "mincut bipartitioning

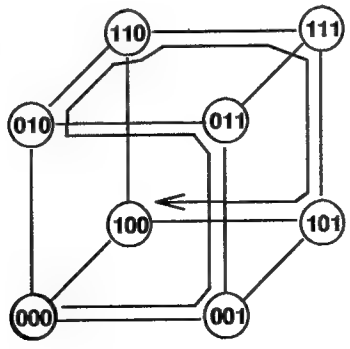

Figure 7: Gray code order traversal of processors in a 3-dimensional hypercube; the key to the operation of the original greedy mapper.

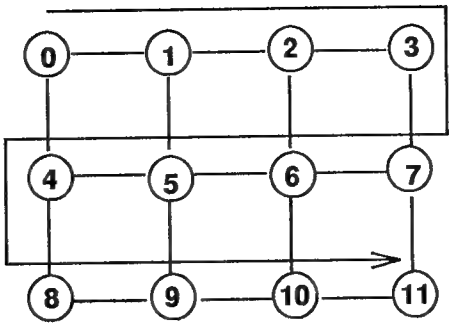

Figure 8: "Snake order" traversal of processors in 4×3 mesh; the key modification made to the greedy mapper for use with mesh topologies.

mapper," which can characterized as a hypercube clustering algorithm. The main idea of this technique is to use a number of recursive calls to a clustering function equal to the dimension of the hypercube. Because the dimension of the hypercube is $\log_2 N$, where N is the number of processors, for the case of the mesh, the key change was to use $\lfloor \log_2 N \rfloor$ recursive calls of the clustering function. This relatively simple modification resulted in reasonable mapping solutions when applied to a mesh topology.

The "hypersphere mapper" algorithm represents the mapping algorithm for which the most extensive changes were made. The key ideas above in the original version of hypersphere mapper, which was designed for hypercubes, were as follows:

- approximate discrete hypercube with continuous hypersphere;
- formulate continuous average distance objective function;
- minimize continuous objective using nonlinear programming techniques; and
- discretize continuous solution onto discrete hypercube.

A detailed description of the hypersphere algorithm has been published in [1].

For using this approach for a mesh, the first and last key ideas were modified. In particular, the mesh is approximated as a continuos plane. Thus, the key ideas in the modified version of this approach are as follows:

- approximate discrete 2-D mesh with continuous 2-D plane;
- formulate continuous average distance objective function;
- minimize continuous objective using nonlinear programming techniques; and
- discretize continuous solution onto discrete 2-D mesh.

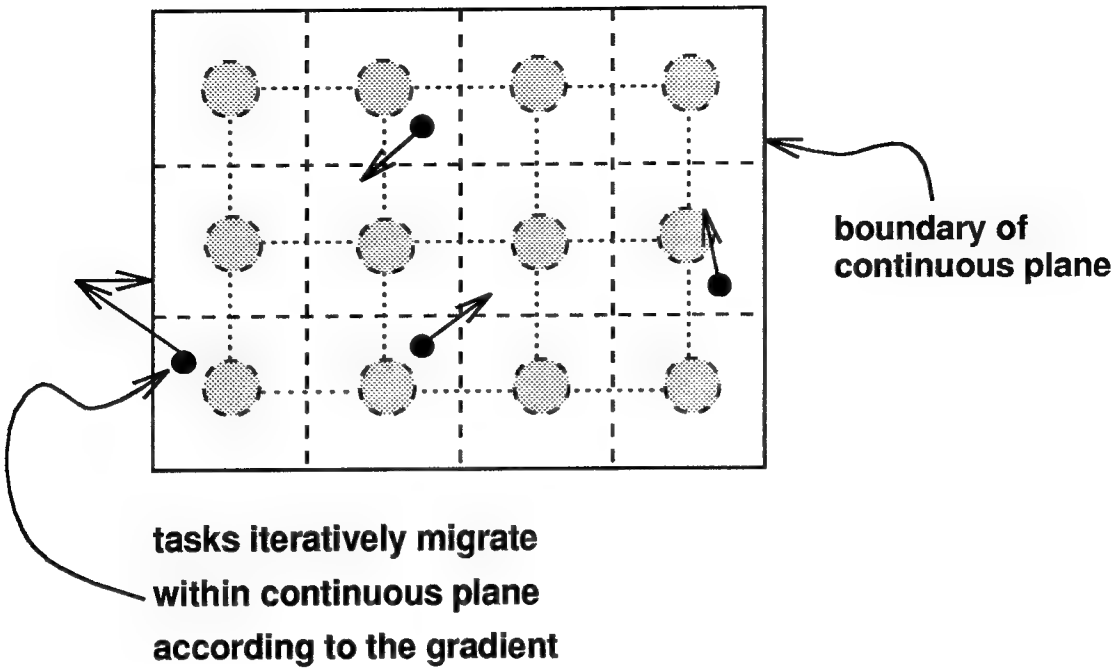

Figure 9: Geometric view of the modification made to the original hypersphere mapper for use with meshes.

A geometric view of this approach is illustrated in Fig. 9. The small solid circles represent subtasks whose "locations" are initially defined with respect to the continuous 2-D plane. The subtasks iteratively migrate around the continuous plane according to the gradient of the continuous objective function. After convergence, the continuous locations of the subtasks are converted to processor numbers based on the cellular decomposition shown (the physical mesh architecture is shown "below" the continuous plan in dashed lines).

Again, not all of the details required to convert the original mappers (for hypercubes) to appropriate mappers for meshes has been given in this section. However, the three conversions overviewed briefly here are representative of the types of changes that had to be made. The source code for the mapping algorithms is well-documented, and the interested reader should consult these listings for more details.

5 Routing Algorithm Modifications

Two routing algorithms are available within OMARS: multiflo router and default router. The multiflo router [4, 3] is an optimal routing algorithm that aims to minimize the link contention of the target architecture. Thus, the multiflo router produces a set of routes (one for each communicating source-destination pair) that tend to minimize the maximum link contention in the architecture. There are two input files required for the multiflo router: (1) the network

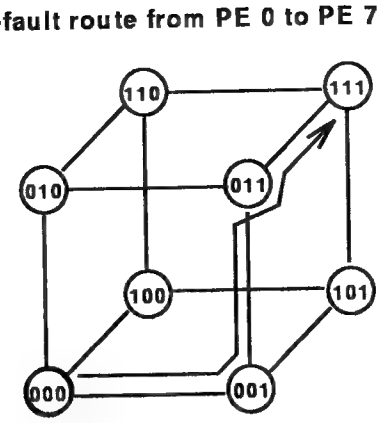

Figure 10: Example of the default route from processor 0 to processor 7 in a 3-dimensional hypercube.

topology file and (2) the communication pattern file. Thus, to implement multiflo router for a given configuration of a mesh, we wrote code that generated the corresponding network topology file the multiflo router. This was a fairly easy modification given that the code had already been written in the original version of OMARS for generating a network topology file for the hypercube architecture. No modification to the original code for generating the communication pattern file was required.

The default router implements the routing used by the target architecture. For example, the default route for a 3-dimensional hypercube from processor 0 (000) to processor 7 (111) is shown in Fig. 10. Note that the default route is defined by correcting bits of the destination in a "right to left" order.

The default router on the Paragon is called an "X-Y" routing scheme. As shown in Fig. 11, the route first moves in the horizontal direction (i.e., "X" direction) and then in the vertical direction (i.e., "Y" direction). Code was written to define the required communication links for any possible source-destination pair in an arbitrary mesh. This addition completed the functionality of the default router for the Paragon.

6 An Application Study on the nCUBE 2

6.1 Overview

The objective of the study described in this section is to demonstrate the effect that mapping has on a parallel application task. The selected parallel algorithm is the FFT, which is used as a "building block" in several real-time applications such as SAR processing and frequency-ramp filtering. The executing timing experiments were conducted on the nCUBE 2.

Timings associated with 1,000 randomly generated mappings were collected, and statisti-

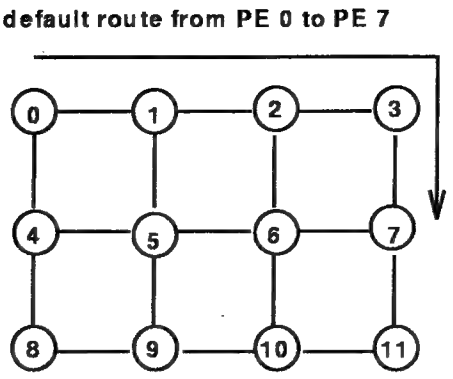

Figure 11: Example of the default route from processor 0 to processor 7 in a mesh.

cal analysis—which quantifies correlation between measured communication time and three different mapping metrics—is presented. Two of the metrics considered, average distance and maximum link utilization factor, are metrics that are used by OMARS for evaluating mapping and routing choices (they are displayed in the architecture graph window). The application program considered for this study is a radix-2 decimation-in-time fast Fourier transform (FFT) algorithm [9].

6.2 The FFT Algorithm

In general, a $P = 2^p$ point FFT algorithm requires a total of p stages of computation, labeled 0, ..., $p-1$. Let $d(q)$ and $D(q)$ denote the values of the qth input and output points of the FFT algorithm, respectively, $0 \leq q \leq 2^p - 1$. Let the binary representation of label q be denoted by $q = (q_{p-1} \ldots q_0)$, and let $\text{cube}_i(q) = (q_{p-1} \ldots \bar{q}_i \ldots q_0)$ [10]. At stage i of computation, the current value of the qth point is multiplied by an appropriate weighting factor, and the result is added to the current value of point number $\text{cube}_{n-i}(q)$ (see Fig. 12). Refer to [9] for precise definitions of the weighting factors.

For this study, the input data set was decomposed into P/N contiguous blocks (where $N = 2^n$ is the number of nodes of the nCUBE 2 that were used), and P was chosen to be larger than N. Thus, the set of points initially allocated to subtask j, $0 \leq j \leq N-1$, is given by $\{d(q) : (\frac{P}{N})j \leq q \leq (\frac{P}{N})(j+1) - 1\}$. Based on this data decomposition, the first $n = \log(N)$ stages of the FFT require communication among subtasks. The last $p-n$ stages require only local data for each subtask. The communication pattern for the first n stages is as follows: at stage i, $0 \leq i \leq n-1$, subtask j sends data to subtask $\text{cube}_{n-i}(j)$. At each of these stages, every subtask sends a message to and receives a message from one other task. For example, for $n = 2$ and $p = 3$, subtask 0 sends data points $d(0)$ and $d(1)$ together as one message to subtask 2, at stage 0. This is illustrated in Fig. 13.

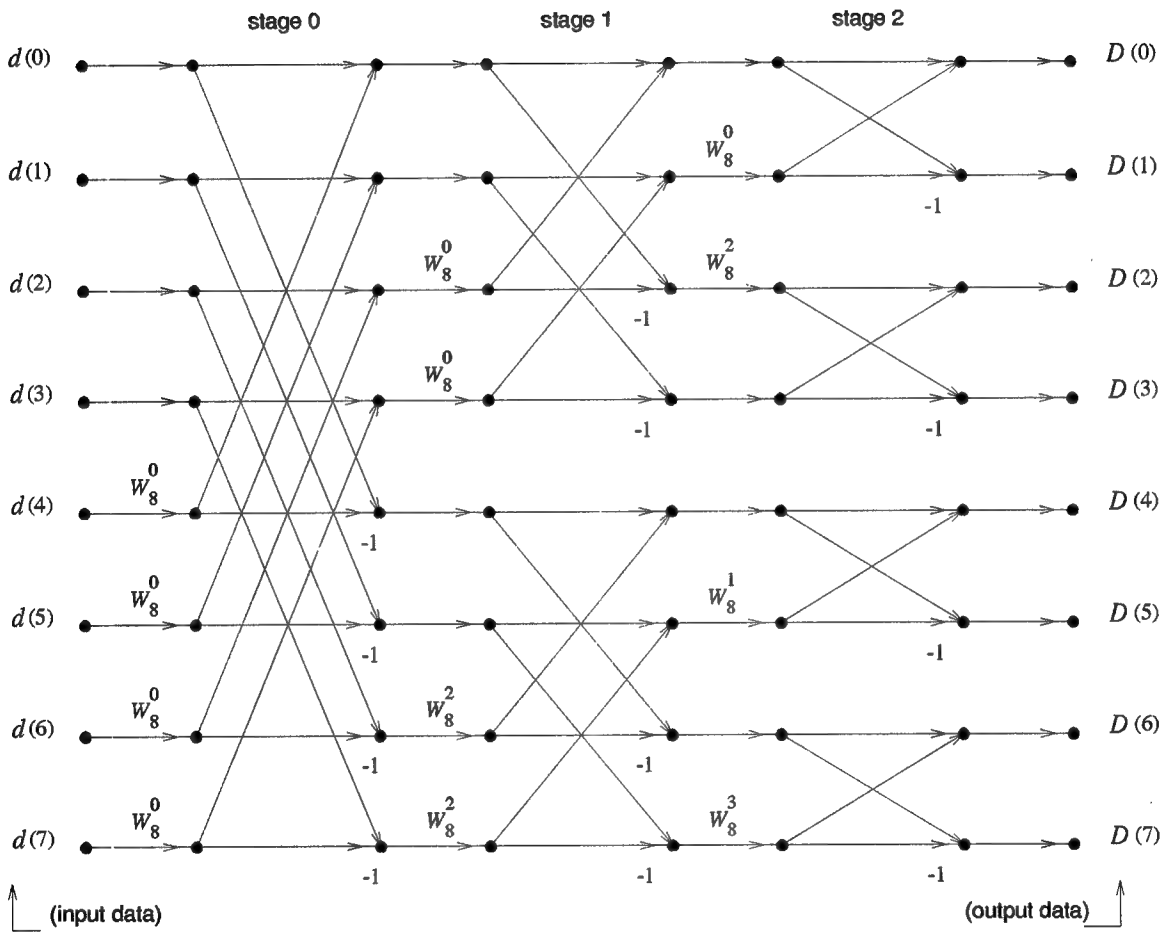

Figure 12: Data flow graph for a $P = 8$ point decimation-in-time FFT algorithm.

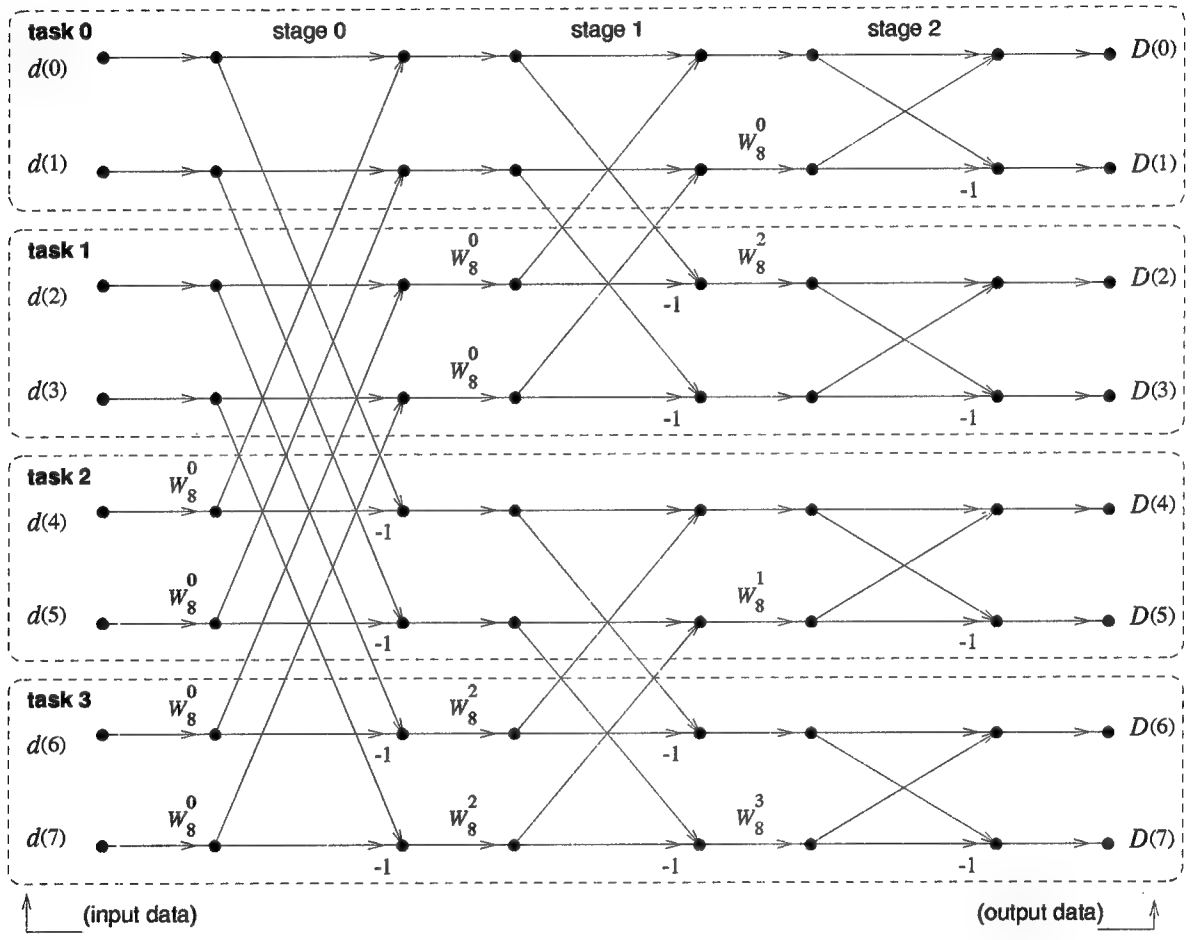

Figure 13: Decomposition of a P point FFT algorithm into N subtasks ($P = 8$, $N = 4$).

6.3 Metrics for Quantifying Mapping Quality

Three metrics were used to characterize mappings for the parallel FFT algorithm: average distance, $\underline{\alpha}$; maximum link utilization, $\underline{\beta}$; and sum of the per stage maximum link utilization, $\underline{\gamma}$. Average distance (α) refers to the average distance between all pairs of subtasks that communicate over the execution of the entire program (i.e., over all n stages of communication). The maximum link utilization (β) indicates the maximum number of times any link is used during the execution of the entire program. To define the sum of the per stage maximum link utilization, it is necessary to first define the stage i maximum link utilization, which is the maximum number of times any link is used during stage i. Therefore, the sum of the per stage maximum link utilization (γ) is the sum of all stage i maximum link utilizations, $0 \leq i \leq n-1$.

The previously defined N subtasks associated with the parallel FFT algorithm can be mapped onto the N nodes of the nCUBE 2 in many different ways (i.e., $N!$ different ways). An obvious choice is to map subtask j to node j. For this particular mapping, at every stage, a subtask will communicate with another that is one link away. Therefore, the value of the average distance metric for this mapping is $\alpha = 1$. As would be expected, this mapping performs very well when implemented because there is no contention for communication resources. The "optimality" of this mapping is well documented in the literature (e.g., see [5]). Recall, however, that the purpose of this experimental study is not to identify a single "good" mapping, but to quantify the "relative goodness" of any given mapping based on the associated values of the aforementioned metrics.

6.4 Timing Results

The purpose of this part of the study is to quantify the "quality" of a given mapping by correlating computed values of the metrics α, β, and γ with measured communication times from the nCUBE 2. Numerous mappings (1,000) were obtained by randomly assigning one subtask per node. A partition of $N = 2^4$ nodes was used in this study, and the number of input data points for the FFT was $P = 2^{12}$. Thus, each subtask was assigned a group of $2^8 = 256$ points. The arithmetic operations (multiplication and addition) required by the FFT algorithm were not included, as the focus of this study was to investigate the impact of different mappings on the communication time of the FFT program. These arithmetic operations would introduce only a constant factor in the overall execution time, and eliminating them minimized the time required to conduct these experimental studies on the nCUBE 2.

Scatter plots of α, β, and γ versus communication times are shown in Figs. 14, 15, and 16, respectively. For all three figures, solid lines are drawn through the average of the measured

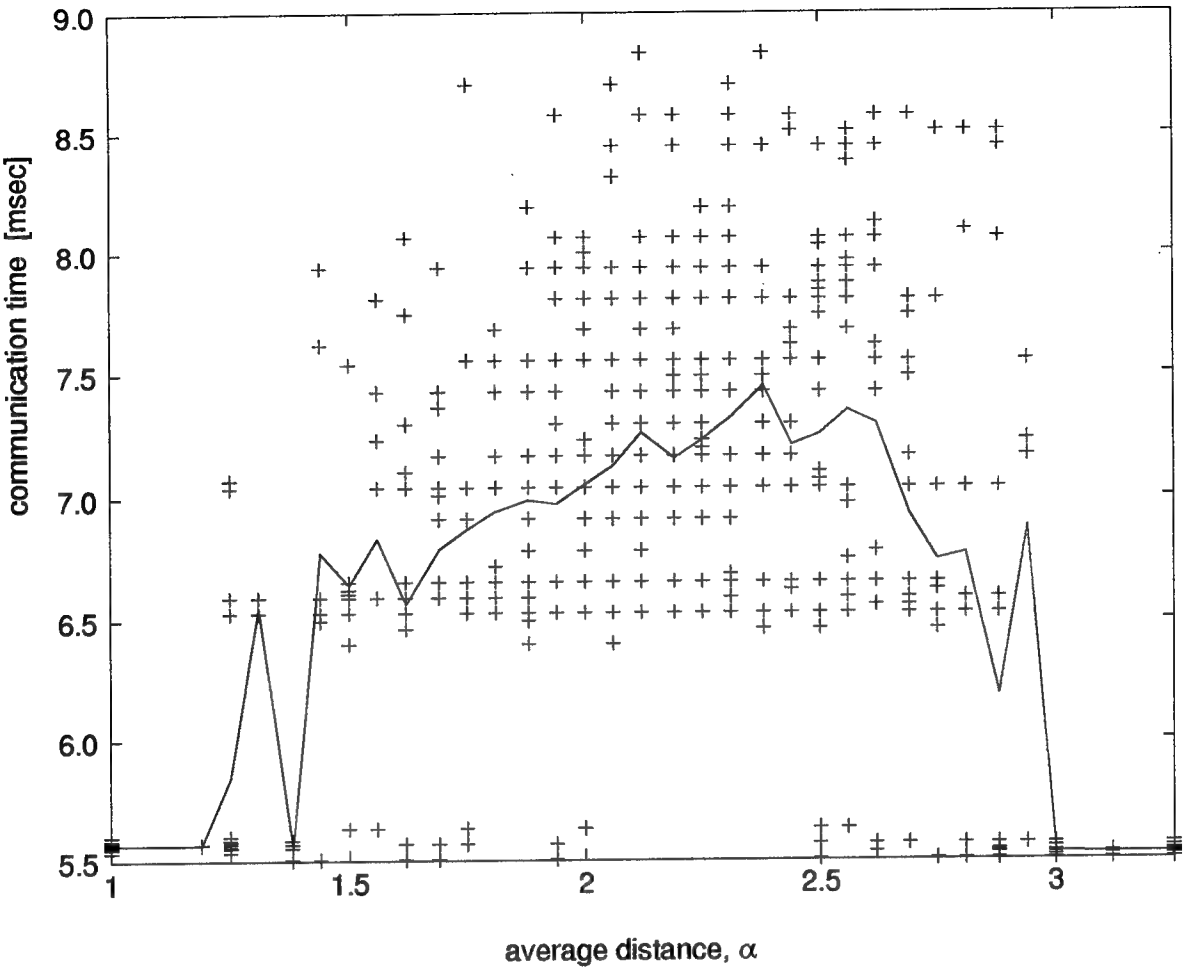

Figure 14: Scatter plot of α versus measured communication time.

communication times for each observed metric value. The correlation coefficients between metrics and communication times are shown in Table 1.

Of the three metrics considered, note that α is the poorest overall predictor of communication time. In fact, communication time is virtually uncorrelated with the value of this metric. From Fig. 14, observe the gap between "good" mappings, which have communication times around 5.5 msecs, and "bad" mappings, which have communication times over 6.5 msecs. Although it is not surprising that all mappings with a value of $\alpha = 1$ are good mappings, it is somewhat surprising that there is a significant number of mappings with $\alpha > 1$ that are also good mappings. In fact, all mappings with $\alpha \geq 3$ are good mappings.

To understand these results, observe from Fig. 16 that every good mapping (as defined previously from Fig. 14) has the same minimal possible value of γ (i.e., $\gamma = n = 4$). Thus, the value of α may be high for a given mapping, however, if its associated value of γ is minimal,

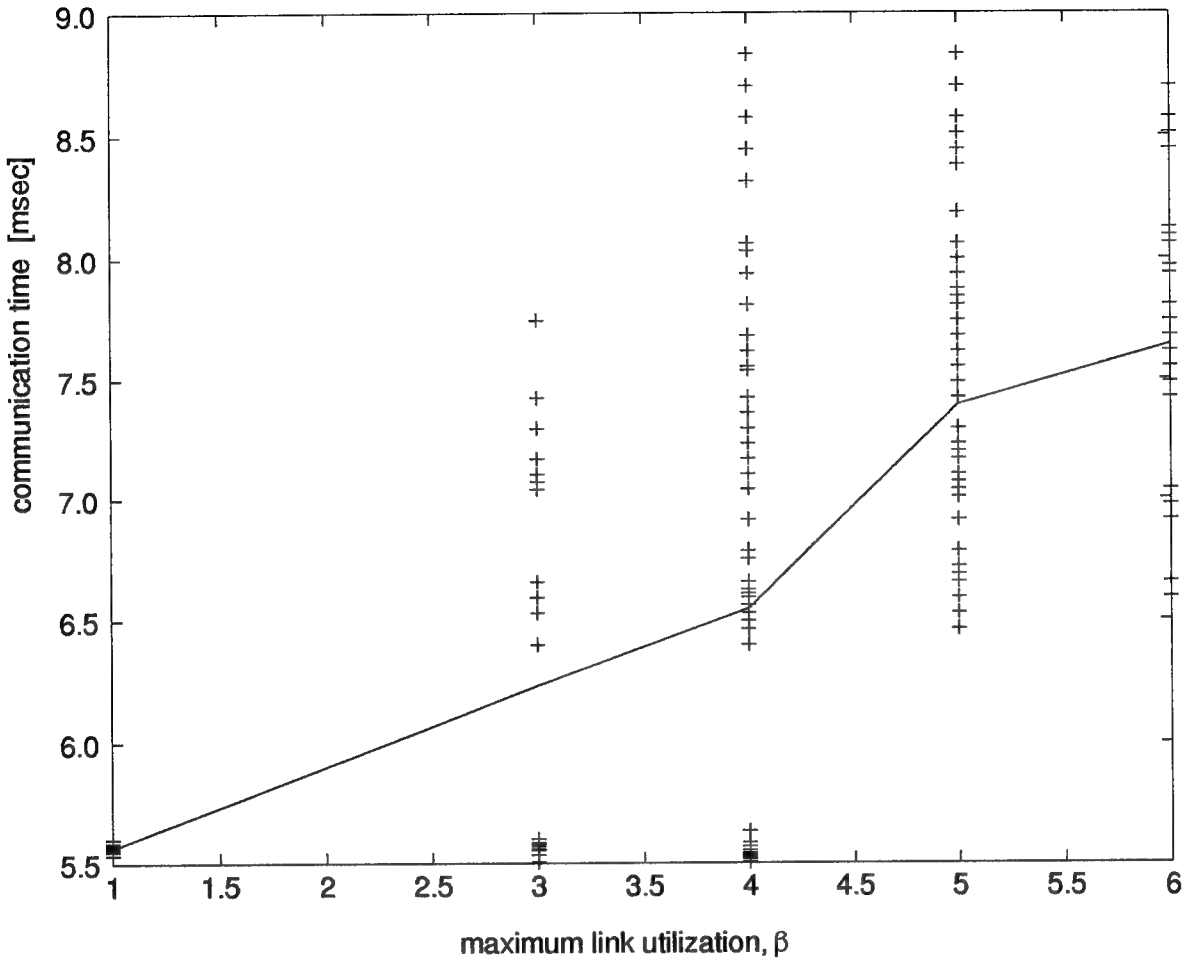

Figure 15: Scatter plot of β versus measured communication time.

Table 1: Correlation coefficients between metrics and measured communication times.

metric	correlation coefficient
α	0.0258
β	0.5858
γ	0.8349

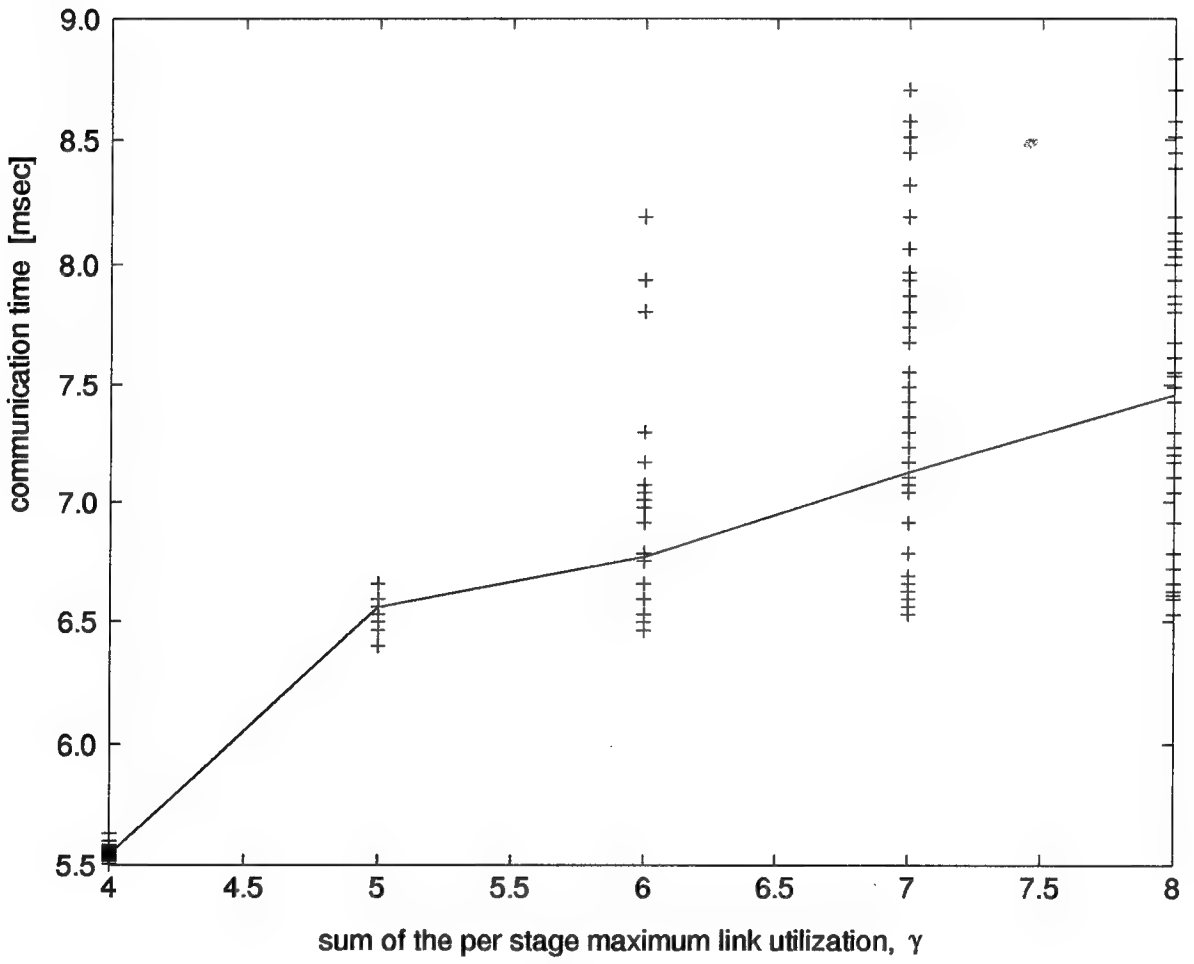

Figure 16: Scatter plot of γ versus measured communication time.

then the mapping is good. This result does make intuitive sense when one considers how communication paths are established on the nCUBE 2. In particular, because the paths are established using fast routing hardware, communication times can be relatively independent of the value of α, provided that there is no link contention. (A more detailed explanation of how paths are established through the hypercube network of the nCUBE 2 is described in Section 7.) The minimum value of γ ($\gamma = 4$ in this case), represents mappings in which each link in the hypercube is used by at most one path during any given stage of communication.

The fact that all generated mappings that had a value of $\alpha \geq 3$ had corresponding values of $\gamma = 4$ is a result of a graph-theoretic relationship between the communication pattern of the FFT program and the topology of the hypercube. Thus, for the FFT program, "spreading out" the subtasks as far as possible—as measured by the value of α—actually minimized link contention for each stage of communication. However, one would not necessarily expect this to be the case for general applications and/or different network topologies.

The metric β correlates reasonably well with communication time, i.e., better than α but not as well as γ. Note from Fig. 15 that there are some good mappings for which the corresponding value of β is not minimal (i.e., there are good mappings that have values $\beta = 3$ and $\beta = 4$). These represent mappings for which at least one link of the hypercube is used by three or four paths over the execution of the entire program, but these same links are used only once during each phase of communication. Thus, there is actually no link contention at any instant in time for these mappings. Clearly, β does not discriminate between spatial and temporal contention, but γ does.

Although γ is the best of the three metrics considered for the FFT program, it is also the least generally applicable across various application domains. In particular, γ cannot be directly computed for applications in which there are no "stages of communication." In contrast, the metric β (which does not have the temporal fidelity of γ) can be applied to any application program, and thus may be the metric of choice in general. One possible way to utilize the metric γ for general applications would be to capture, during the tracing of a general program, timing information between communications. In this way, it may be possible to detect "pseudo-stages of communication," and thus compute γ accordingly. However, extra time would be required to analyze and cluster this traced timing information into pseudo-stages of communication.

7 Evaluation of the Alternate Routing Capability of the nCUBE 2

In its default (i.e., normal) operating mode, the nCUBE 2 system establishes routes through its hypercube network according to the default routing scheme as defined in Section 5. Special communication hardware at each node, called the Network Communication Unit (NCU) [8], provides support for this routing function. To send data from one node to another on the nCUBE 2, the library function `nwrite()` is used. This construct gathers information needed to build the actual message that is sent through the network.

Messages are composed of two parts: the header and the body. The header is the first part of the message and includes information such as the source and destination node numbers, the message type, and the total length of the message. The second part of the message is the body, which is the actual data being sent. The `nwrite()` function processes and assembles the information for the message, and this information is passed to the operating system (OS) using a trap. Code within the OS writes the message to a queue that is readable by the NCU. The NCU takes the message header from the queue, examines the destination node number, calculates the next node in the path (by comparing the destination node number with the node number of the NCU), and sends the message through the appropriate communication channel to the next node along the route.

In addition to default routing, the NCU does provide support for establishing alternate routes through the hypercube interconnection network. This is accomplished by utilizing a special forwarding mechanism, which is enabled by setting a "forwarding bit" within the address field of the header (the destination node number also resides in the address field). To establish an alternate route, the address of each node (except the source) in the route must be listed in a sequence of address fields within the header, and the forwarding bit associated with each of these intermediate address fields must be set, with the exception of the last address, which is the final destination.

When a message header arrives at a node, the NCU checks the status of the forwarding bit associated with the first address field in the header.

- If this forwarding bit is set and the address in this field matches that of the NCU, then the NCU assumes that it is actually an intermediate "forwarding node" (not the destination), and the next address field is used to define the destination (the current address field is discarded). Based on the value of this new destination address, the message is appropriately routed through the network to the next node in the path.

- If the forwarding bit within the first address field is not set upon arrival at an NCU,

then the NCU compares its own address against the destination address in this field, and appropriately sends the message to the next node in the path (or transfers the body of the message to the current processor's memory if it has arrived at its final destination).

To prevent improper use and/or invalid routes from being requested, the nCUBE 2 system software is intentionally designed to prevent the portion of the address field where the forwarding bit resides from being accessed (either directly or indirectly) by an application programmer. Only the OS of the nCUBE 2, called Vertex, can set this bit. For the purposes of this experimental study, a licensing agreement was granted to Purdue from nCUBE Corporation, which provided us with the source code of Vertex. We were able to modify the Vertex code so that forwarding address fields could be inserted into the message header through custom library calls made from the application source code.

A simple experiment was conducted to illustrate the potential advantage of implementing alternate routes. Eight nodes of the nCUBE 2 (i.e., a 3-dimensional hypercube) were used for this study, and a simple program was written to send two messages between two distinct pairs of nodes. The first message, denoted as message 0, contained 10K bytes of data and was sent from node 1 (001) to node 3 (011). The second message, denoted as message 1, contained only 8 bytes of data and was sent from node 0 (000) to node 7 (111), refer to Fig. 17. The single link used to route the large message 0 from node 1 to node 3 is an intermediate link along the default route from node 0 to node 7. The measured time to transmit message 1 under different conditions are provided in Table 2.

The first column of times in Table 2 correspond to the case where the large message 0 is not transmitted at all (i.e., the corresponding `nwrite()` and `nread()` commands were not included in the source code). The second column of times in the table are for the case where the large message 0 is transmitted (i.e., the link between nodes 1 and 3 is congested). In this case, if the default route for message 1 is used, then the corresponding communication time is large because the path for message 1 cannot be established until after the large message 0 has completed transmission. However, by using the alternate route, the congested link is avoided and a communication time is achieved that is comparable to the case in which message 0 is not sent. This alternate routing is a solution from the modified multiflo routing algorithm of OMARS.

The communication time using the alternate route is 4 μsecs longer than that of the default route with no congestion because the required message header is longer. In particular, two extra address fields (four bytes each) were inserted and transmitted with the message header in order to establish the alternate route. However, the advantage in using the alternate route outweighs the disadvantage of this small extra overhead. The time associated with

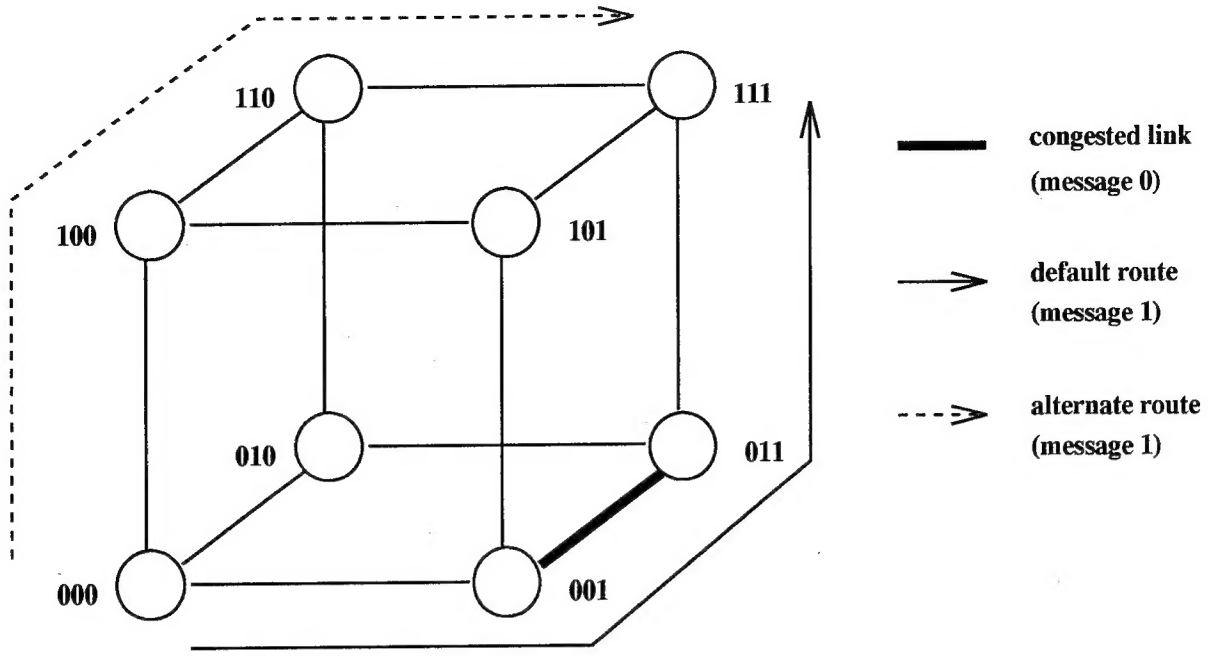

Figure 17: Overview of the scenario used for the alternate routing experiment. The alternate routing shown is a solution from the modified multiflo routing algorithm of OMARS.

Table 2: Measured communication times (in μsecs) for message 1 of Fig. 9.

OS/(selected route)	message 0 not sent	message 0 sent
original Vertex (default route)	190	1885
customized Vertex (alternate route)	194	194

using the default route when message 0 is sent (i.e., 1885 μsecs) is proportional to the size of message 0; 10K bytes was used for this experiment to ensure that the link from node 1 to 3 was severely congested. Using different sizes for message 0 would produce correspondingly different values for this time.

8 Summary and Future Work

The development of OMARS is an ongoing effort. The primary goal of OMARS is to provide software engineers with some tools for addressing relatively low-level issues (i.e., mapping and routing) that arise when developing large software applications on parallel processing platforms. The functionality of OMARS was enhanced in this effort by including a new target architecture: the Intel Paragon. The importance of making effective mapping and routing

decisions was illustrated based on experiments from the nCUBE 2. Plans are currently being made to introduce OMARS, on an experimental basis, to students in a parallel programming class, as a means of getting user feedback. Plans beyond this initial release are to eventually make the tool available to the general public.

Acknowledgments

A number of people have contributed to the OMARS project over the past several years. Without their expert skills in concept design and programming, OMARS would have never become a reality. The contributors to OMARS include: John K. Antonio, Eduardo Asbun, Farhan A. Baqai, Loretta S. Ellwood, Yan A. Li, Richard C. Metzger, Chester A. Wright, Jr., and Olivia K. Wu.

The author would like to extend a special thanks to the program manager for the OMARS project, Richard C. Metzger of Rome Laboratory. His numerous technical contributions and motivation for this project over the years were the keys to its success. The assistance of Chester A. Wright, Jr. of Rome Laboratory is also very much appreciated; especially with regard to Unix questions and preparing the final demonstration of OMARS.

Finally, the author thanks David Wiener of nCUBE Corporation for arranging the licensing agreement for the Vertex OS.

References

[1] J. K. Antonio and R. C. Metzger, "Hypersphere mapper: a nonlinear programming approach to the hypercube embedding problem," *J. Parallel and Distributed Computing*, vol. 19, no. 3, Nov. 1993, pp. 262–270.

[2] J. K. Antonio, "Software Techniques for Balancing Computation & Communication in Parallel Systems," Rome Laboratory, Air Force Materiel Command, Griffiss Air Force Base, NY, Final Technical Report No. RL-TR-94-98, July 1994.

[3] J. K. Antonio, W. K. Tsai, and G. M. Huang, "Time complexity of a path formulated optimal routing algorithm," *IEEE Trans. Automatic Control*, vol. 39, no. 9, Sept. 1994, pp. 1839–1844.

[4] D. P. Bertsekas, B. Gendron, and W. K. Tsai, "Implementation of an optimal multicommodity network flow algorithm based on gradient projection and a path flow formulation," MIT, Cambridge, MA, LIDS Rep. No. LIDS-P-1364, Feb. 1984.

[5] A. Gupta and V. Kumar, "The scalability of FFT on parallel computers," *IEEE Trans. Parallel and Distributed Systems*, vol. 4, no. 8, Aug. 1993, pp.922–932.

[6] V. Kumar, A. Grama, A. Gupta, and G. Karypis, *Introduction to Parallel Computing: Design and Analysis of Algorithms*, Benjamin/Cummings, Redwood City, CA, 1994.

[7] R. C. Metzger, L. S. Auvil, C. A. Wright, Jr., J. K. Antonio, Y. A. Li, O. K. Wu, and E. Asbun, "OMARS: Optimal Mapping Alternate Routing System," *Proceedings of the Parallel Systems Fair of the 8th International Parallel Processing Symposium*, Apr. 1994, pp. 11–20.

[8] nCUBE Corporation, *nCUBE 2 Processor Manual*, Order # 101636, nCUBE Corporation, Dec. 1990.

[9] J. G. Proakis and D. G. Manolakis, *Digital Signal Processing: Principles, Algorithms, and Applications*, Second Edition, MacMillan, New York, NY, 1992.

[10] H. J. Siegel, *Interconnection Networks for Large-Scale Parallel Processing*, Second Edition, McGraw-Hill, New York, NY, 1990.

*U.S. GOVERNMENT PRINTING OFFICE: 1997-509-127-47114

MISSION OF ROME LABORATORY

Mission. The mission of Rome Laboratory is to advance the science and technologies of command, control, communications and intelligence and to transition them into systems to meet customer needs. To achieve this, Rome Lab:

a. Conducts vigorous research, development and test programs in all applicable technologies;

b. Transitions technology to current and future systems to improve operational capability, readiness, and supportability;

c. Provides a full range of technical support to Air Force Material Command product centers and other Air Force organizations;

d. Promotes transfer of technology to the private sector;

e. Maintains leading edge technological expertise in the areas of surveillance, communications, command and control, intelligence, reliability science, electro-magnetic technology, photonics, signal processing, and computational science.

The thrust areas of technical competence include: Surveillance, Communications, Command and Control, Intelligence, Signal Processing, Computer Science and Technology, Electromagnetic Technology, Photonics and Reliability Sciences.